

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

## REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS  
BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: International Business Machines Corporation, IBM Development System for the Ada Language, MVS Ada Compiler, Version 2.1.1, IBM 4381 (Host & Target), 890420W1.10074		5. TYPE OF REPORT & PERIOD COVERED 20 Apr. 1989 to 20 Apr. 1990
7. AUTHOR(s) Wright-Patterson AFB Dayton, OH, USA		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson AFB Dayton, OH, USA		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081		12. REPORT DATE
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Wright-Patterson AFB Dayton, OH, USA		13. NUMBER OF PAGES
		15. SECURITY CLASS (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A
16. DISTRIBUTION STATEMENT (of this Report)  Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20 if different from Report)  UNCLASSIFIED		
18. SUPPLEMENTARY NOTES		
19. KEYWORDS (Continue on reverse side if necessary and identify by block number)  Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number)  International Business Machines Corporation, IBM Development System for the Ada Language, MVS Ada Compiler, Version 2.1.1, Wright-Patterson AFB, IBM 4381 under MVS/XA, Release 2.7 (Host & Target), ACVC 1.10.		

DTIC  
ELECTE  
AUG 4 1989

S

A

I

30 8 03 038

DD FORM 1473  
1 JAN 73EDITION OF 1 NOV 65 IS OBSOLETE  
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD-A211'056

AVF Control Number: AVF-VSR-261.0789  
89-01-26-TEL

Ada COMPILER  
VALIDATION SUMMARY REPORT:  
Certificate Number: 890420W1.10074  
International Business Machines Corporation  
IBM Development System for the Ada Language  
MVS Ada Compiler, Version 2.1.1  
IBM 4381

Completion of On-Site Testing:  
20 April 1989

Prepared By:  
Ada Validation Facility  
ASD/SCEL  
Wright-Patterson AFB OH 45433-6503

Prepared For:  
Ada Joint Program Office  
United States Department of Defense  
Washington DC 20301-3081



2  
A 21

Ada Compiler Validation Summary Report:

Compiler Name: IBM Development System for the Ada Language,  
MVS Ada Compiler, Version 2.1.1


Certificate Number: 890420W1.10074

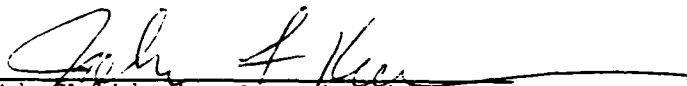
Host: IBM 4381 under  
MVS/XA, Release 2.7


Target: IBM 4381 under  
MVS/XA, Release 2.7

Testing Completed 20 April 1989 Using ACVC 1.10

This report has been reviewed and is approved.

  
\_\_\_\_\_  
Ada Validation Facility  
Steve P. Wilson  
Technical Director  
ASD/SCEL  
Wright-Patterson AFB OH 45433-6503

  
\_\_\_\_\_  
Ada Validation Organization  
Dr. John F. Kramer  
Institute for Defense Analyses  
Alexandria VA 22311

  
for \_\_\_\_\_ Deputy Director  
Ada Joint Program Office  
Dr. John Solomond  
Director  
Department of Defense  
Washington DC 20301

Ada Compiler Validation Summary Report:

Compiler Name: IBM Development System for the Ada Language,  
MVS Ada Compiler, Version 2.1.1


Certificate Number: 890420W1.10074

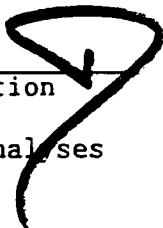
Host: IBM 4381 under  
MVS/XA, Release 2.7

Target: IBM 4381 under  
MVS/XA, Release 2.7

Testing Completed 20 April 1989 Using ACVC 1.10

This report has been reviewed and is approved.

  
\_\_\_\_\_  
Ada Validation Facility  
Steve P. Wilson  
Technical Director  
ASD/SCEL  
Wright-Patterson AFB OH 45433-6503

  
\_\_\_\_\_  
Ada Validation Organization  
Dr. John F. Kramer  
Institute for Defense Analyses  
Alexandria VA 22311

\_\_\_\_\_  
Ada Joint Program Office  
Dr. John Solomond  
Director  
Department of Defense  
Washington DC 20301

## TABLE OF CONTENTS

CHAPTER 1	INTRODUCTION	
1.1	PURPOSE OF THIS VALIDATION SUMMARY REPORT . . . .	1-2
1.2	USE OF THIS VALIDATION SUMMARY REPORT . . . . .	1-2
1.3	REFERENCES. . . . .	1-3
1.4	DEFINITION OF TERMS . . . . .	1-3
1.5	ACVC TEST CLASSES . . . . .	1-4
CHAPTER 2	CONFIGURATION INFORMATION	
2.1	CONFIGURATION TESTED. . . . .	2-1
2.2	IMPLEMENTATION CHARACTERISTICS. . . . .	2-2
CHAPTER 3	TEST INFORMATION	
3.1	TEST RESULTS. . . . .	3-1
3.2	SUMMARY OF TEST RESULTS BY CLASS. . . . .	3-1
3.3	SUMMARY OF TEST RESULTS BY CHAPTER. . . . .	3-2
3.4	WITHDRAWN TESTS . . . . .	3-2
3.5	INAPPLICABLE TESTS. . . . .	3-2
3.6	TEST, PROCESSING, AND EVALUATION MODIFICATIONS. .	3-6
3.7	ADDITIONAL TESTING INFORMATION. . . . .	3-6
3.7.1	Prevalidation . . . . .	3-6
3.7.2	Test Method . . . . .	3-6
3.7.3	Test Site . . . . .	3-8
APPENDIX A	DECLARATION OF CONFORMANCE	
APPENDIX B	APPENDIX F OF THE Ada STANDARD	
APPENDIX C	TEST PARAMETERS	
APPENDIX D	WITHDRAWN TESTS	

## CHAPTER 1

### INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation-dependent but is permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

## INTRODUCTION

### 1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- . To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- . To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard
- . To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc. under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 20 April 1989 at San Diego CA.

### 1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C.#552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse  
Ada Joint Program Office  
OUSDRE  
The Pentagon, Rm 3D-139 (Fern Street)  
Washington DC 20301-3081

or from:

Ada Validation Facility  
ASD/SCEL  
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization  
Institute for Defense Analyses  
1801 North Beauregard Street  
Alexandria VA 22311

### 1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.
3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.
4. Ada Compiler Validation Capability User's Guide, December 1986.

### 1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the <u>Ada Compiler Validation Procedures and Guidelines</u> .
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including



## INTRODUCTION

cross-compilers, translators, and interpreters.

Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

### 1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors because of the way in which a program library is used at link time.

Class A tests ensure the successful compilation of legal Ada programs with certain language constructs which cannot be verified at compile time. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every

## INTRODUCTION

illegal construct that it contains is detected by the compiler.

Class C tests check the run time system to ensure that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Class E tests are expected to execute successfully and check implementation-dependent options and resolutions of ambiguities in the Ada Standard. Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated. In some cases, an implementation may legitimately detect errors during compilation of the test.

Two library units, the package REPORT and the procedure CHECK\_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK\_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK\_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of each test in the ACVC follows conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate

## INTRODUCTION

tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated. A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2  
CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: IBM Development System for the Ada Language,  
MVS Ada Compiler, Version 2.1.1

ACVC Version: 1.10

Certificate Number: 890420W1.10074

Host Computer:

Machine:	IBM 4381
Operating System:	MVS/XA Release 2.7
Memory Size:	32 Megabytes

Target Computer:

Machine:	IBM 4381
Operating System:	MVS/XA Release 2.7
Memory Size:	32 Megabytes

## CONFIGURATION INFORMATION

### 2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

#### a. Capacities.

- (1) The compiler correctly processes a compilation containing 723 variables in the same declarative part. (See test D29002K.)
- (2) The compiler correctly processes tests containing loop statements nested to 65 levels. (See tests D55A03A..H (8 tests).)
- (3) The compiler correctly processes tests containing block statements nested to 65 levels. (See test D56001B.)
- (4) The compiler correctly processes tests containing recursive procedures separately compiled as subunits nested to 10 levels. (See tests D64005E..G (3 tests).)

#### b. Predefined types.

- (1) This implementation supports the additional predefined types `SHORT_INTEGER` and `LONG_FLOAT` in package `STANDARD`. (See tests B86001T..Z (7 tests).)

#### c. Expression evaluation.

The order in which expressions are evaluated and the time at which constraints are checked are not defined by the language. While the ACVC tests do not specifically attempt to determine the order of evaluation of expressions, test results indicate the following:

- (1) Some of the default initialization expressions for record components are evaluated before any value is checked for membership in a component's subtype. (See test C32117A.)
- (2) Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)
- (3) This implementation uses no extra bits for extra precision and uses no extra bits for extra range. (See test C35903A.)

## CONFIGURATION INFORMATION

- (4) Sometimes `NUMERIC_ERROR` is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)
- (5) `NUMERIC_ERROR` is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)
- (6) Underflow is not gradual. (See tests C45524A..Z.)

### d. Rounding.

The method by which values are rounded in type conversions is not defined by the language. While the ACVC tests do not specifically attempt to determine the method of rounding, the test results indicate the following:

- (1) The method used for rounding to integer is round away from zero. (See tests C46012A..Z.)
- (2) The method used for rounding to longest integer is round away from zero. (See tests C46012A..Z.)
- (3) The method used for rounding to integer in static universal real expressions is round away from zero. (See test C4A014A.)

### e. Array types.

An implementation is allowed to raise `NUMERIC_ERROR` or `CONSTRAINT_ERROR` for an array having a `'LENGTH'` that exceeds `STANDARD.INTEGER'LAST` and/or `SYSTEM.MAX_INT`.

For this implementation:

- (1) Declaration of an array type or subtype declaration with more than `SYSTEM.MAX_INT` components raises `NUMERIC_ERROR` sometimes. (See test C36003A.)
- (2) `NUMERIC_ERROR` is raised when a null array type with `INTEGER'LAST + 2` components is declared. (See test C36202A.)
- (3) `NUMERIC_ERROR` is raised when a null array type with `SYSTEM.MAX_INT + 2` components is declared. (See test C36202B.)
- (4) A packed `BOOLEAN` array having a `'LENGTH'` exceeding `INTEGER'LAST` raises `NUMERIC_ERROR` when the array type is declared. (See test C52103X.)

## CONFIGURATION INFORMATION

- (5) A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC\_ERROR when the array type is declared. (See test C52104Y.)
- (6) A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC\_ERROR or CONSTRAINT\_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC\_ERROR when the array type is declared. (See test E52103Y.)
- (7) In assigning one-dimensional array types, the expression is evaluated in its entirety before CONSTRAINT\_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)
- (8) In assigning two-dimensional array types, the expression is not evaluated in its entirety before CONSTRAINT\_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

### f. Discriminated types.

- (1) In assigning record types with discriminants, the expression is evaluated in its entirety before CONSTRAINT\_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

### g. Aggregates.

- (1) In the evaluation of a multi-dimensional aggregate, index subtype checks are made as choices are evaluated. (See tests C43207A and C43207B.)
- (2) In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)
- (3) CONSTRAINT\_ERROR is raised after all choices are evaluated when a bound in a non-null range of a non-null aggregate does not belong to an index subtype. (See test E43211B.)

### h. Pragmas.

- (1) The pragma INLINE is not supported for functions or procedures. (See tests LA3004A..B, EA3004C..D, and CA3004E..F.)

## CONFIGURATION INFORMATION

### i. Generics

- (1) Generic library subprogram specifications and bodies can be compiled in separate compilations. (See test CA1012A.)
- (2) Generic non-library subprogram bodies cannot be compiled in separate compilations from their stubs. (See test CA2009F.)
- (3) Generic library package specifications and bodies can be compiled in separate compilations. (See tests BC3204C and BC3205D.)
- (4) Generic non-library package bodies as subunits cannot be compiled in separate compilations. (See test CA2009C.)
- (5) Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

### j. Input and output

- (1) The package `SEQUENTIAL_IO` cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)
- (2) The package `DIRECT_IO` cannot be instantiated with unconstrained array types or record types with discriminants without defaults. (See tests AE2101E, EE2401D, and EE2401G.)
- (3) Modes `IN FILE` and `OUT FILE` are supported for `SEQUENTIAL_IO`. (See tests CE2102D..E, CE2102N, and CE2102P.)
- (4) Modes `IN FILE`, `OUT FILE`, and `INOUT FILE` are supported for `DIRECT_IO`. (See tests CE2102F, CE2102I..J, CE2102R, CE2102T, and CE2102V.)
- (5) Modes `IN FILE` and `OUT FILE` are supported for text files. (See tests CE3102E and CE3102I..K.)
- (6) `RESET` and `DELETE` operations are supported for `SEQUENTIAL_IO`. (See tests CE2102G and CE2102X.)
- (7) `RESET` and `DELETE` operations are supported for `DIRECT_IO`. (See tests CE2102K and CE2102Y.)
- (8) `RESET` and `DELETE` operations are supported for text files. (See tests CE3102F..G, CE3104C, CE3110A, and CE3114A.)
- (9) Overwriting to a sequential file does not truncate the file. (See test CE2208B.)



## CONFIGURATION INFORMATION

- (10) Temporary sequential files are given names and deleted when closed. (See test CE2108A.)
- (11) Temporary direct files are given names and deleted when closed. (See test CE2108C.)
- (12) Temporary text files are given names and deleted when closed. (See test CE3112A.)
- (13) More than one internal file can be associated with each external file for sequential files when reading only. (See tests CE2107A..E, CE2102L, CE2110B, and CE2111D.)
- (14) More than one internal file can be associated with each external file for direct files when reading only. (See tests CE2107F..H (3 tests), CE2110D, and CE2111H.)
- (15) More than one internal file can be associated with each external file for text files when reading only. (See tests CE3111A..E, CE3114B, and CE3115A.)

## CHAPTER 3

### TEST INFORMATION

#### 3.1 TEST RESULTS

Version 1.10 of the ACVC comprises 3717 tests. When this compiler was tested, 43 tests had been withdrawn because of test errors. The AVF determined that 318 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 201 executable tests that use floating-point precision exceeding that supported by the implementation. Modifications to the code, processing, or grading for 6 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

#### 3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	127	1131	2016	16	22	44	3356
Inapplicable	2	7	300	1	6	2	318
Withdrawn	1	2	34	0	6	0	43
TOTAL	130	1140	2350	17	34	46	3717

## TEST INFORMATION

### 3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	198	573	544	244	171	99	160	333	129	36	252	341	276	3356
Inappl	14	76	136	4	1	0	6	0	8	0	0	28	45	318
Wdrn	1	1	0	0	0	0	0	1	0	0	1	35	4	43
TOTAL	213	650	680	248	172	99	166	334	137	36	253	404	325	3717

### 3.4 WITHDRAWN TESTS

The following 43 tests were withdrawn from ACVC Version 1.10 at the time of this validation:

E28005C	A39005G	B97102E	BC3009B	CD2A62D	CD2A63A
CD2A63B	CD2A63C	CD2A63D	CD2A66A	CD2A66B	CD2A66C
CD2A66D	CD2A73A	CD2A73B	CD2A73C	CD2A73D	CD2A76A
CD2A76B	CD2A76C	CD2A76D	CD2A81G	CD2A83G	CD2A84M
CD2A84N	CD2B15C	CD2D11B	CD5007B	CD50110	ED7004B
ED7005C	ED7005D	ED7006C	ED7006D	CD7105A	CD7203B
CD7204B	CD7205C	CD7205D	CE2107I	CE3111C	CE3301A
CE3411B					

See Appendix D for the reason that each of these tests was withdrawn.

### 3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 318 tests were inapplicable for the reasons indicated:

- a. The following 201 tests are not applicable because they have floating-point type declarations requiring more digits than `SYSTEM.MAX_DIGITS`:

C24113L..Y	C35705L..Y	C35706L..Y	C35707L..Y
C35708L..Y	C35802L..Z	C45241L..Y	C45321L..Y
C45421L..Y	C45521L..Z	C45524L..Z	C45621L..Z
C45641L..Y	C46012L..Z		

# TEST INFORMATION

- b. C35508I, C35508J, C35508M, and C35508N are not applicable because they include enumeration representation clauses for BOOLEAN types in which the representation values are other than (FALSE => 0, TRUE => 1). Under the terms of AI-00325, this implementation is not required to support such representation clauses.
- c. C35702A and B86001T are not applicable because this implementation supports no predefined type SHORT\_FLOAT.
- d. The following 16 tests are not applicable because this implementation does not support a predefined type LONG\_INTEGER:

C45231C	C45304C	C45502C	C45503C	C45504C
C45504F	C45611C	C45613C	C45614C	C45631C
C45632C	B52004D	C55B07A	B55B09C	B86001W
CD7101F				

- e. C45231D, B86001X, and CD7101G are not applicable because this implementation does not support any predefined integer type with a name other than INTEGER, LONG\_INTEGER, or SHORT\_INTEGER.
- f. C45531M..P (4 tests) and C45532M..P (4 tests) are not applicable because the value of SYSTEM.MAX\_MANTISSA is less than 47.
- g. C52008B is not applicable because this implementation does not support a record type with four discriminants of type integer having default values. The size of this object exceeds the maximum object size of this implementation and NUMERIC\_ERROR is raised.
- h. D64005G is not applicable because this implementation does not support nesting 17 levels of recursive procedure calls.
- i. C86001F is not applicable because, for this implementation, the package TEXT\_IO is dependent upon package SYSTEM. These tests recompile package SYSTEM, making package TEXT\_IO, and hence package REPORT, obsolete.
- j. B86001Y is not applicable because this implementation supports no predefined fixed-point type other than DURATION.
- k. B86001Z is not applicable because this implementation supports no predefined floating-point type with a name other than FLOAT, LONG\_FLOAT, or SHORT\_FLOAT.
- l. CA2009C is not applicable because this implementation does not permit compilation of generic non-library package bodies as subunits in separate files from their stubs.
- m. CA2009F is not applicable because this implementation does not permit compilation of generic non-library subprogram bodies as subunits in separate files from their stubs.
- n. LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F are not

## TEST INFORMATION

applicable because this implementation does not support pragma `INLINE`.

- o. CD1009C, CD2A41A..B (2 tests), CD2A41E, and CD2A42A..J (10 tests) are not applicable because this implementation does not support size clauses for floating point types.
- p. CD2A61I and CD2A61J are not applicable because this implementation does not support size clauses for array types, which imply compression, with component types of composite or floating point types. This implementation requires an explicit size clause on the component type.
- q. CD2A84B..I (8 tests) and CD2A84K..L (2 tests) are not applicable because this implementation does not support size clauses for access types.
- r. AE2101C, EE2201D, and EE2201E use instantiations of package `SEQUENTIAL_IO` with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.
- s. AE2101H, EE2401D, and EE2401G use instantiations of package `DIRECT_IO` with unconstrained array types and record types with discriminants without defaults. These instantiations are rejected by this compiler.
- t. CE2102D is inapplicable because this implementation supports `CREATE` with `IN_FILE` mode for `SEQUENTIAL_IO`.
- u. CE2102E is inapplicable because this implementation supports `CREATE` with `OUT_FILE` mode for `SEQUENTIAL_IO`.
- v. CE2102F is inapplicable because this implementation supports `CREATE` with `INOUT_FILE` mode for `DIRECT_IO`.
- w. CE2102I is inapplicable because this implementation supports `CREATE` with `IN_FILE` mode for `DIRECT_IO`.
- x. CE2102J is inapplicable because this implementation supports `CREATE` with `OUT_FILE` mode for `DIRECT_IO`.
- y. CE2102N is inapplicable because this implementation supports `OPEN` with `IN_FILE` mode for `SEQUENTIAL_IO`.
- z. CE2102O is inapplicable because this implementation supports `RESET` with `IN_FILE` mode for `SEQUENTIAL_IO`.
- aa. CE2102P is inapplicable because this implementation supports `OPEN` with `OUT_FILE` mode for `SEQUENTIAL_IO`.
- ab. CE2102Q is inapplicable because this implementation supports `RESET` with `OUT_FILE` mode for `SEQUENTIAL_IO`.
- ac. CE2102R is inapplicable because this implementation supports `OPEN` with `INOUT_FILE` mode for `DIRECT_IO`.

## TEST INFORMATION

- ad. CE2102S is inapplicable because this implementation supports RESET with INOUT\_FILE mode for DIRECT\_IO.
- ae. CE2102T is inapplicable because this implementation supports OPEN with IN\_FILE mode for DIRECT\_IO.
- af. CE2102U is inapplicable because this implementation supports RESET with IN\_FILE mode for DIRECT\_IO.
- ag. CE2102V is inapplicable because this implementation supports open with OUT\_FILE mode for DIRECT\_IO.
- ah. CE2102W is inapplicable because this implementation supports RESET with OUT\_FILE mode for DIRECT\_IO.
- ai. CE2107B..E (4 tests), CE2107L, CE2110B, and CE2111D are not applicable because multiple internal files cannot be associated with the same external file when one or more files is writing for sequential files. The proper exception is raised when multiple access is attempted.
- aj. CE2107G..H (2 tests), CE2110D, and CE2111H are not applicable because multiple internal files cannot be associated with the same external file when one or more files is writing for direct files. The proper exception is raised when multiple access is attempted.
- ak. CE2201G is inapplicable because this implementation does not support CREATE with OUT\_FILE mode for SEQUENTIAL IO where the item type is a record type with variants and discriminants having default values.
- al. CE2401H is inapplicable because this implementation does not support CREATE with INOUT\_FILE mode for unconstrained record with default discriminants.
- am. CE3102E is inapplicable because this implementation supports CREATE with IN\_FILE mode for text files.
- an. CE3102F is inapplicable because this implementation supports RESET for text files.
- ao. CE3102G is inapplicable because this implementation supports deletion of an external file for text files.
- ap. CE3102I is inapplicable because this implementation supports CREATE with OUT\_FILE mode for text files.
- aq. CE3102J is inapplicable because this implementation supports OPEN with IN\_FILE mode for text files.
- ar. CE3102K is inapplicable because this implementation supports OPEN with OUT\_FILE mode for text files.

## TEST INFORMATION

as. CE3111B, CE3111D..E (2 tests), CE3114B, and CE3115A are not applicable because multiple internal files cannot be associated with the same external file when one or more files is writing for text files. The proper exception is raised when multiple access is attempted.

### 3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behavior. Modifications are made by the AVF in cases where legitimate implementation behavior prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behavior that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 6 tests.

The following tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

BA3006A      BA3006B      BA3007B      BA3008A      BA3008B      BA3013A

### 3.7 ADDITIONAL TESTING INFORMATION

#### 3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.10 produced by the IBM Development System for the Ada Language, MVS Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

#### 3.7.2 Test Method

Testing of the IBM Development System for the Ada Language, MVS Ada Compiler using ACVC Version 1.10 was conducted on-site by a validation team from the AVF. The configuration in which the testing was performed is described by the following designations of hardware and software components:

Host computer:	IBM 4381
Host operating system:	MVS/XA, Release 2.7
Target computer:	IBM 4381
Target operating system:	MVS/XA, Release 2.7
Compiler:	IBM Development System for the Ada Language, MVS Ada Compiler, Version 2.1.1

## TEST INFORMATION

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the prevalidation testing were included in their modified form on the magnetic tape.

The contents of the magnetic tape were ported onto the host computer after first being loaded to a VM/CMS computer system. No conversion or alteration of contents occurred.

After the test files were loaded to disk, the full set of tests was compiled, linked, and all executable tests were run on the IBM 4381. Results were printed from the host computer.

The compiler was tested using command scripts provided by International Business Machines Corporation and reviewed by the validation team. The compiler was tested using all default option settings except for the following:

OPTION	EFFECT
-----	-----
ERROR(LIST)	Creates a listing file only when errors are encountered. The file contains compile-time error messages interspersed with source code.
LIST(ERRI)	Produces a compilation source listing. Semantic errors, syntax errors, and warnings are interspersed.
RUN(TEXT)	Causes the program to load and execute. It is assumed that the program display the results on the console. The output of the entire compilation and execution is copied to a dataset. This dataset is examined to determine whether the program was executed successfully.

Tests were compiled, linked, and executed (as appropriate) using a single computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

### 3.7.3 Test Site

Testing was conducted at San Diego CA and was completed on 20 April 1989.



APPENDIX A

DECLARATION OF CONFORMANCE

International Business Machines Corporation has submitted the following Declaration of Conformance concerning the IBM Development System for the Ada Language, MVS Ada Compiler.

## DECLARATION OF CONFORMANCE

Compiler Implementor: TeleSoft

Ada Validation Facility: ASD/SCEL, Wright-Patterson AFB, OH 45433-6503

Ada Compiler Validation Capability (ACVC) Version: 1.10

### Base Configuration

Base Compiler Name: IBM Development System for the Ada Language,  
MVS Ada Compiler, Version 2.1.1

Host Architecture ISA: IBM 4381

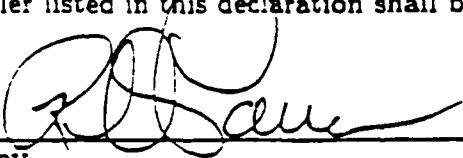
Operating System: MVS/XA Release 2.7

Target Architecture ISA: IBM 4381

Operating System: MVS/XA Release 2.7

### Implementor's Declaration

I, the undersigned, representing TeleSoft have implemented no deliberate extensions to the Ada Language Standard ANSI/MIL-STD-1815A in the compiler listed in this declaration. I declare that International Business Machines Corporation is the owner of record of the object code of the Ada language compiler listed above and, as such, is responsible for maintaining said compiler in conformance to ANSI/MIL-STD-1815A. All certificates and registrations for the Ada language compiler listed in this declaration shall be made only in the owner's corporate name.



TeleSoft

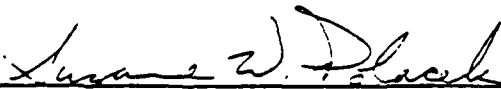
Raymond A. Parra, Director, Contracts & Legal

Date:

2/21/89

### Owner's Declaration

I, the undersigned, representing International Business Machines Corporation take full responsibility for implementation and maintenance of the Ada compiler listed above, and agree to the public disclosure of the final Validation Summary Report. I declare that all of the Ada language compilers listed, and their host/target performance are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.



International Business Machines Corporation

S. W. Polacek, Manager of Advanced Language Products

Date:

2/21/89

## APPENDIX B

### APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of the Ada Standard, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of the IBM Development System for the Ada Language, MVS Ada Compiler, Version 2.1.1, as described in this Appendix, are provided by TeleSoft. Unless specifically noted otherwise, references in this Appendix are to compiler documentation and not to this report. Implementation-specific portions of the package STANDARD, which are not a part of Appendix F, are:

package STANDARD is

...

type INTEGER is range -2\_147\_483\_648 .. 2147\_483\_647;

type SHORT\_INTEGER is range -32\_768 .. 32\_767;

type FLOAT is digits 6 range -7.23701E+75 .. 7.23701E+75;

type LONG\_FLOAT is digits 15 range -7.23700557733225E+75 ..

7.23700557733225E+75;

type DURATION is delta 2#1.0#E-14 range -86400.0 .. 86400.0;

...

end STANDARD;

## APPENDIX F OF THE LANGUAGE REFERENCE MANUAL

The Ada language definition allows for certain target dependencies in a controlled manner. This section, called Appendix F as prescribed in the LRM, describes implementation-dependent characteristics of the IBM Development System for the Ada Language Release 2.1.1 running under CMS or MVS.

### 1. Implementation-Defined Pragmas

PRAGMA INTERFACE( Assembly, <subroutine\_name> );

PRAGMA INTERFACE( Non\_XA\_Assembly, <subroutine\_name> );

PRAGMA INTERFACE( Fortran, <subroutine\_name> );

PRAGMA SUPPRESS\_ALL;

to cause Pragma SUPPRESS to be invoked simultaneously for all the following condition\_names: access\_check, discriminant\_check, index\_check, length\_check, division\_check, elaboration\_check, and storage\_check.

PRAGMA COMMENT (string\_literal);

embeds string\_literal into object code

PRAGMA IMAGES (enumeration\_type, <immediate>|<deferred>);

generates a table of images for the enumeration type. deferred causes the table to be generated only if the enumeration type is used in a compilation unit

PRAGMA LINKNAME (<pragma INTERFACE subprogram name>, <linkname>);

when used in association with pragma INTERFACE, will provide access to any routine whose name can be specified by an Ada string literal.

\*PRAGMA MVSTASK (priority);

to specify the relative urgency of each MVS task created.

\*PRAGMA ALLOCATION\_DATA

(<access\_type>,  
<residence\_mode>,  
<allocation\_duration>,  
<subpool\_number>,  
<discrete\_user\_data>);

to associate MVS virtual storage attributes with an Ada access type.

**Note** that PRAGMA MVSTASK and PRAGMA ALLOCATION\_DATA are effective only when compiling for an MVS target. Both pragmas require that an MVS runtime be present.

### 2. Implementation-Defined Attributes

There are no implementation-defined attributes.

### 3. Package SYSTEM

The current specification of package SYSTEM is provided below.

PACKAGE System IS

TYPE Address is Access Integer;  
TYPE Name IS (MC68000, ANUYK44, IBM370);

System\_Name : CONSTANT name := IBM370;

Storage\_Unit : CONSTANT := 8;  
Memory\_Size : CONSTANT := 2\*\*24-1;

-- System-Dependent Named Numbers:

Min\_Int : CONSTANT := -(2 \*\* 31);  
Max\_Int : CONSTANT := (2 \*\* 31) - 1;  
Max\_Digits : CONSTANT := 15;  
Max\_Mantissa : CONSTANT := 31;  
Fine\_Delta : CONSTANT := 1.0 / (2 \*\* Max\_Mantissa );  
Tick : CONSTANT := 1.0 / (10 \*\* 6);

-- Other System-Dependent Declarations

TYPE Subprogram\_Value is record  
    Entry\_Point\_Address : Address;  
    Display : Address;  
end record;  
SUBTYPE Priority IS Integer RANGE -255 .. 255;

end SYSTEM;

### 4. Representation Clauses

This implementation supports address, length, enumeration, and record representation clauses with the following exceptions:

Address clauses are not supported for package, for entry, for tasktype, for subprograms.

Enumeration clauses are not supported for boolean representation clauses.

The size in bits of representation specified records is rounded up to the next highest multiple of 8, meaning that the object of a representation specified record with 25 bits will actually occupy 32 bits, and, if the record is used as a component of another representation specified record, 32 bits must be reserved for it.

Non-supported clauses are rejected at compile time.

## **5. Implementation-Generated Names**

There are no implementation-generated names denoting implementation-dependent components. Names generated by the compiler shall not interfere with programmer-defined names.

## **6. Address Clause Expression Interpretation**

Expressions that appear in Address clauses are interpreted as virtual memory addresses.

## **7. Unchecked Conversion Restrictions**

Unchecked conversions are allowed between types (or subtypes) T1 and T2 provided that:

- (1) They have the same static size.
- (2) They are not private.

## **8. Implementation-Dependent Characteristics of the I/O Packages**

- Sequential\_IO, Direct\_IO, and Text\_IO are supported.
- Low\_Level\_IO is not supported.
- Unconstrained array types and unconstrained types with discriminants may not be instantiated for I/O.
- File names follow the conventions and restrictions of the target operating system.
- In Text\_IO, the type Field is defined as follows: subtype Field is integer range 0..1000;
- In Text\_IO, the type Count is defined as follows: type Count is range 0..2\_147\_483\_646;

# ATTACHMENT F PARAMETERS USED IN .TST TESTS (MACRO SUBSTITUTIONS)

```

$MAX IN LENGTH: 200
$BIG_ID1: STRING(1..200) := (1..199 => 'A', 200 => '1')
$BIG_ID2: STRING(1..200) := (1..199 => 'A', 200 => '2')
$BIG_ID3: STRING(1..200) := (1..100 => 'A', 101 => '3', 102..200 => 'A')
$BIG_ID4: STRING(1..200) := (1..100 => 'A', 101 => '4', 102..200 => 'A')
$BIG_STRING1: STRING(1..102) := (1 => '"', 2..101 => 'A', 102 => '"')
$BIG_STRING2: STRING(1..102) := (1 => '"', 2..100 => 'A', 101..102 => '1"')
$MAX_STRING_LITERAL: STRING(1..200) := (1 => '"', 2..199 => 'A', 200 => '"')
$NEG_BASED_INT: 16#FFFFFFFE#
$BIG_INT_LIT: STRING(1..200) := (1..197 => '0', 198..200 => "298")
$BIG_REAL_LIT: STRING(1..200) := (1..194 => '0', 195..200 => "69.0E1")
$MAX_LEN_INT_BASED_LITERAL: STRING(1..200) := (1..2 => "2:", 3..197 =>
    '0', 198..200 => "11:")
$MAX_LEN_REAL_BASED_LITERAL: STRING(1..200) := (1..3 => "16:", 4..196 =>
    '0', 197..200 => "F.E:")
$BLANKS: STRING(1..180) := (1..180 => ' ')
$MAX DIGITS: 15
$NAME: NO SUCH TYPE AVAILABLE
$FLOAT NAME: NO SUCH TYPE
$FIXED NAME: NO SUCH TYPE
$INTEGER_FIRST: -2 147 483 648
$INTEGER_LAST: 2 147 483 647
$INTEGER_LAST_PLUS_1: 2 147 483 648
$MIN_INT: -2 147 483 648
$MAX_INT: 2 147 483 647
$MAX_INT_PLUS_1: 2 147 483 648
$LESS_THAN_DURATION: -86 401.0
$GREATER_THAN_DURATION: 86 401.0
$LESS_THAN_DURATION_BASE_FIRST: -131 073.0
$GREATER_THAN_DURATION_BASE_LAST: 131 073.0
$COUNT_LAST: 2 147 483 646
$FIELD_LAST: 1 000
$ILLEGAL_EXTERNAL_FILE_NAME1: "BADCHAR*"/%"
$ILLEGAL_EXTERNAL_FILE_NAME2: "/NONAME/DIRECTORY"
$ACC SIZE: 32
$TASK SIZE: 32
$MIN_TASK SIZE: 32
$NAME LIST: MC68000, ANUYK44, IBM370
$DEFAULT_SYS_NAME: IBM370
$NEW_SYS_NAME: IBM370
$DEFAULT_STOR UNIT: 8
$NEW_STOR UNIT: 8
$DEFAULT_MEM SIZE: 16777215
$NEW_MEM SIZE: 16777215
$LOW_PRIORITY: -255

```

SHIGH PRIORITY: 255  
SMANTISSA\_DOC: 31  
\$DELTA\_DOC: 2#1.0#E-31  
\$TICK: 0.000001



# APPENDIX C TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
\$ACC_SIZE An integer literal whose value is the number of bits sufficient to hold any value of an access type.	32
\$BIG_ID1 An identifier the size of the maximum input line length which is identical to \$BIG_ID2 except for the last character.	(1..199=>'A',200=>'1')
\$BIG_ID2 An identifier the size of the maximum input line length which is identical to \$BIG_ID1 except for the last character.	(1..199=>'A',200=>'2')
\$BIG_ID3 An identifier the size of the maximum input line length which is identical to \$BIG_ID4 except for a character near the middle.	(1..100=>'A',101=>'3',102..200=>'A')

## TEST PARAMETERS

Name and Meaning	Value
<b>\$BIG_ID4</b> An identifier the size of the maximum input line length which is identical to \$BIG_ID3 except for a character near the middle.	(1..100=>'A',101=>'4',102..200=>'A')
<b>\$BIG_INT_LIT</b> An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..197=>'0',198..200=>"298")
<b>\$BIG_REAL_LIT</b> A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	(1..194=>'0',195..200=>"690.0")
<b>\$BIG_STRING1</b> A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	(1=>'"',2..101=>'A',102=>' "')
<b>\$BIG_STRING2</b> A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	(1=>'"',2..100=>'A',101=>'1',102=>' "')
<b>\$BLANKS</b> A sequence of blanks twenty characters less than the size of the maximum line length.	(1..180=>' ')
<b>\$COUNT_LAST</b> A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483645
<b>\$DEFAULT_MEM_SIZE</b> An integer literal whose value is SYSTEM.MEMORY_SIZE.	16777215
<b>\$DEFAULT_STOR_UNIT</b> An integer literal whose value is SYSTEM.STORAGE_UNIT.	8

# TEST PARAMETERS

Name and Meaning	Value
<b>SDEFAULT_SYS NAME</b> The value of the constant SYSTEM.SYSTEM_NAME.	ibm370
<b>SDELTA_DOC</b> A real literal whose value is SYSTEM.FINE_DELTA.	0.000000000931322574615478515625
<b>SFIELD_LAST</b> A universal integer literal whose value is TEXT_IO.FIELD'LAST.	1000
<b>SFIXED_NAME</b> The name of a predefined fixed-point type other than DURATION.	NO_SUCH_TYPE
<b>SFLOAT_NAME</b> The name of a predefined floating-point type other than FLOAT, SHORT_FLOAT, or LONG_FLOAT.	NO_SUCH_TYPE
<b>SGREATER THAN DURATION</b> A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	86401.0
<b>SGREATER THAN DURATION BASE LAST</b> A universal real literal that is greater than DURATION'BASE'LAST.	131073.0
<b>SHIGH PRIORITY</b> An integer literal whose value is the upper bound of the range for the subtype SYSTEM.PRIORITY.	255
<b>SILLEGAL EXTERNAL FILE NAME1</b> An external file name which contains invalid characters.	"BADCHAR*~/%"
<b>SILLEGAL EXTERNAL FILE NAME2</b> An external file name which is too long.	"/NONAME/DIRECTORY"
<b>SINTEGER FIRST</b> A universal integer literal whose value is INTEGER'FIRST.	-2147483648

# TEST PARAMETERS

<u>Name and Meaning</u>	<u>Value</u>
\$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	2147483647
\$INTEGER_LAST PLUS 1 A universal integer literal whose value is INTEGER'LAST + 1.	2147483648
\$LESS THAN DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-86401.0
\$LESS THAN DURATION BASE FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-131073.0
\$LOW PRIORITY An integer literal whose value is the lower bound of the range for the subtype SYSTEM.PRIORITY.	-255
\$MANTISSA DOC An integer literal whose value is SYSTEM.MAX_MANTISSA.	31
\$MAX DIGITS Maximum digits supported for floating-point types.	15
\$MAX IN LEN Maximum input line length permitted by the implementation.	200
\$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
\$MAX_INT PLUS 1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
\$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	(1..2=>"2:",3..197=>'0',198..200=>"11:")

# TEST PARAMETERS

Name and Meaning	Value
<p>\$MAX_LEN_REAL_BASED_LITERAL</p> <p>A universal real based literal whose value is 16:F.E: with enough leading zeroes in the mantissa to be MAX_IN_LEN long.</p>	(1..3=>"16:",4..196=>'0', 197..200=>"F.E:")
<p>\$MAX_STRING_LITERAL</p> <p>A string literal of size MAX_IN_LEN, including the quote characters.</p>	(1=>'"',2..199='A',200=>'")
<p>\$MIN_INT</p> <p>A universal integer literal whose value is SYSTEM.MIN_INT.</p>	-2147483648
<p>\$MIN_TASK_SIZE</p> <p>An integer literal whose value is the number of bits required to hold a task object which has no entries, no declarations, and "NULL;" as the only statement in its body.</p>	32
<p>\$NAME</p> <p>A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER.</p>	NO_SUCH_TYPE_AVAILABLE
<p>\$NAME_LIST</p> <p>A list of enumeration literals in the type SYSTEM.NAME, separated by commas.</p>	mc68000,anuyk44,ibm370
<p>\$NEG_BASED_INT</p> <p>A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.</p>	16#FFFFFFFE#
<p>\$NEW_MEM_SIZE</p> <p>An integer literal whose value is a permitted argument for pragma MEMORY_SIZE, other than \$DEFAULT_MEM_SIZE. If there is no other value, then use \$DEFAULT_MEM_SIZE.</p>	16777215

## TEST PARAMETERS

<u>Name and Meaning</u>	<u>Value</u>
<b>\$NEW STOR UNIT</b> An integer literal whose value is a permitted argument for pragma STORAGE UNIT, other than \$DEFAULT_STOR_UNIT. If there is no other permitted value, then use value of SYSTEM.STORAGE_UNIT.	8
<b>\$NEW SYS NAME</b> A value of the type SYSTEM.NAME, other than \$DEFAULT_SYS_NAME. If there is only one value of that type, then use that value.	mc68000,anuyk44
<b>\$TASK SIZE</b> An integer literal whose value is the number of bits required to hold a task object which has a single entry with one 'IN OUT' parameter.	32
<b>\$TICK</b> A real literal whose value is SYSTEM.TICK.	0.000001

APPENDIX D  
WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 43 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form AI-ddddd is to an Ada Commentary.

- a. E28005C: This test expects that the string "-- TOP OF PAGE. --63" of line 204 will appear at the top of the listing page due to a pragma PAGE in line 203; but line 203 contains text that follows the pragma, and it is this text that must appear at the top of the page.
- b. A39005G: This test unreasonably expects a component clause to pack an array component into a minimum size (line 30).
- c. B97102E: This test contains an unintended illegality: a select statement contains a null statement at the place of a selective wait alternative (line 31).
- d. BC3009B: This test wrongly expects that circular instantiations will be detected in several compilation units even though none of the units is illegal with respect to the units it depends on; by AI-00256, the illegality need not be detected until execution is attempted (line 95).
- e. CD2A62D: This test wrongly requires that an array object's size be no greater than 10 although its subtype's size was specified to be 40 (line 137).
- f. CD2A63A..D, CD2A66A..D, CD2A73A..D, and CD2A76A..D (16 tests): These tests wrongly attempt to check the size of objects of a derived type (for which a 'SIZE length clause is given) by passing them to a derived subprogram (which implicitly converts them to the parent type (Ada standard 3.4:14)). Additionally, they use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.

## WITHDRAWN TESTS

- g. CD2A81G, CD2A83G, CD2A84M..N, and CD50110 (5 tests): These tests assume that dependent tasks will terminate while the main program executes a loop that simply tests for task termination; this is not the case, and the main program may loop indefinitely (lines 74, 85, 86, 96, and 58, respectively).
- h. CD2B15C and CD7205C: These tests expect that a 'STORAGE SIZE length clause provides precise control over the number of designated objects in a collection; the Ada standard 13.2:15 allows that such control must not be expected.
- i. CD2D11B: This test gives a SMALL representation clause for a derived fixed-point type (at line 30) that defines a set of model numbers that are not necessarily represented in the parent type; by Commentary AI-00099, all model numbers of a derived fixed-point type must be representable values of the parent type.
- j. CD5007B: This test wrongly expects an implicitly declared subprogram to be at the address that is specified for an unrelated subprogram (line 303).
- k. ED7004B, ED7005C..D, and ED7006C..D (5 tests): These tests check various aspects of the use of the three SYSTEM pragmas; the AVO withdraws these tests as being inappropriate for validation.
- l. CD7105A: This test requires that successive calls to CALENDAR.CLOCK change by at least SYSTEM.TICK; however, by Commentary AI-00201, it is only the expected frequency of change that must be at least SYSTEM.TICK--particular instances of change may be less (line 29).
- m. CD7203B and CD7204B: These tests use the 'SIZE length clause and attribute, whose interpretation is considered problematic by the WG9 ARG.
- n. CD7205D: This test checks an invalid test objective: it treats the specification of storage to be reserved for a task's activation as though it were like the specification of storage for a collection.
- o. CE2107I: This test requires that objects of two similar scalar types be distinguished when read from a file--DATA\_ERROR is expected to be raised by an attempt to read one object as of the other type. However, it is not clear exactly how the Ada standard 14.2.4:4 is to be interpreted; thus, this test objective is not considered valid (line 90).
- p. CE3111C: This test requires certain behavior, when two files are associated with the same external file, that is not required by the Ada standard.
- q. CE3301A: This test contains several calls to END\_OF\_LINE and END\_OF\_PAGE that have no parameter: these calls were intended to specify a file, not to refer to STANDARD\_INPUT (lines 103, 107, 118,



WITHDRAWN TESTS

132, and 136).

- r. CE3411B: This test requires that a text file's column number be set to COUNT'LAST in order to check that LAYOUT\_ERROR is raised by a subsequent PUT operation. But the former operation will generally raise an exception due to a lack of available disk space, and the test would thus encumber validation testing.